

Suchen

---

- Gegeben ist ein Array oder eine Liste von Werten eines bestimmten Typs, z. B. Zahlen
- Wie lässt sich herausfinden, ob (und wo) ein bestimmter Wert darin vorkommt?
- Lösungsansatz:
  - Durchlaufe das Array/die Liste ein Mal komplett, vergleiche jedes Element mit dem zu suchenden
  - Dies bezeichnet man als **Lineare Suche**

- Implementiere eine Methode `suche(int wert)`, die prüft, ob `wert` in einem Array vorkommt
  - Die Methode soll Teil einer Klasse `SucheArray` sein
  - Das Array mit den zu durchsuchenden Werten soll im Konstruktor erzeugt und mit Zufallszahlen gefüllt werden
  - Die Methode soll -1 zurückgeben, falls `wert` im Array nicht vorhanden ist, ansonsten den Index des ersten Vorkommens

- Implementiere eine Methode `suche(int wert)`, die prüft, ob `wert` in einer einfach verketteten Liste vorkommt
  - Die Methode soll Teil einer Klasse `SucheListe` sein
  - Die verkettete Liste mit den zu durchsuchenden Werten soll im Konstruktor erzeugt und mit Zufallszahlen gefüllt werden
  - Die Methode soll `null` zurückgeben, falls `wert` im Array nicht vorhanden ist, ansonsten eine Referenz auf das erste Listenelement, das den gesuchten Wert enthält

- Oft interessiert man sich dafür, wie lange ein Algorithmus braucht, um ein Problem abhängig von der Eingabegröße zu lösen
- Denkbar: Führe den Algorithmus mehrfach mit unterschiedlichen Eingaben aus, stoppe die Zeit
  - Probleme:
    - Abhängig von der verwendeten Hardware
    - Schwankungen bei den Messergebnissen zu erwarten
- Daher: Theoretische Abschätzung der Laufzeit

- Angenommen, wir haben ein Array/eine Liste mit  $n$  Elementen
- Um darin alle Vorkommen eines bestimmten Werts zu suchen, müssen wir der Reihe nach alle Elemente anschauen und vergleichen
  - Je größer  $n$ , desto länger dauert das
  - Wird  $n$  verdoppelt, so erwarten wir auch eine Verdoppelung der Laufzeit
- Etwas allgemeiner:
  - Die Laufzeit der linearen Suche wächst linear mit der Anzahl der Elemente  $n$
  - Wir schreiben dafür: Die Laufzeit der linearen Suche liegt in  $O(n)$

- $O(f(n))$  beinhaltet alle Funktionen, die “nicht wesentlich schneller” wachsen als die Funktion  $f$ 
  - In unserem Beispiel:  $O(n)$  beinhaltet alle Funktionen, die “nicht wesentlich schneller” wachsen als eine Funktion  $f(n) = n$
- Exakte Definition:

$$O(f(n)) := \{g(n) | \exists c > 0, n_0 \in \mathbb{N} \text{ mit } 0 \leq g(n) \leq c \cdot f(n) \text{ für } n \geq n_0\}$$

(aus Goos, Vorlesungen über Informatik, Band 1)

- Wie gibt man in O-Notation an, wenn eine Funktion quadratisch mit der Eingabegröße wächst?
- Wie gibt man in O-Notation eine Funktion mit konstanter Laufzeit (also unabhängig von der Eingabegröße) an?



- Kann man schneller suchen als in  $O(n)$ ?
  - Ja, wenn die Daten, in denen man sucht, bereits sortiert ist
  - Betrachte z. B. das Nachschlagen eines Worts in einem Wörterbuch/Lexikon

- Betrachte das mittlere Element im sortierten Array (wie nehmen aufsteigende Sortierung an)
  - Entspricht es dem gesuchten Element, so endet die Suche erfolgreich
  - Ist es kleiner als das gesuchte Element, so muss sich das gesuchte Element rechts der Mitte befinden
  - Ist es größer als das gesuchte Element, so muss sich das gesuchte Element links der Mitte befinden
- Verfahre auf der linken bzw. rechten Hälfte genauso
  - Die Suche endet, wenn entweder das Element gefunden wird oder das Array nicht mehr weiter halbiert werden kann

- Implementiere die binäre Suche
  - Verwende ggf. die Vorlage
  - Der Algorithmus kann iterativ oder rekursiv implementiert werden

# Laufzeit der binären Suche

- Die binäre Suche halbiert in jede Schritt das Array, maximal so oft, bis nur noch ein Element übrig bleibt
    - Das Halbieren erfolgt nur durch das Anpassen der linken und rechten Grenze, Aufwand hierfür also  $O(1)$
    - Pro Schritt wird nur das Element in der Mitte des gerade betrachteten Teilarrays verglichen, Aufwand hierfür also  $O(1)$
- ⇒ Der Gesamtaufwand pro Schritt liegt also bei  $O(1)$
- Wie viele Schritte gibt es (wie oft kann man maximal halbieren)?
    - Die Anzahl der Schritte entspricht dem Zweierlogarithmus von  $n$
  - Laufzeit binäre Suche:  $O(\log_2 n)$  bzw.  $O(\log n)$